2023

A Quantum Solution to the Travelling Salesman Problem

iQuHACK 2023

AmplifiQation | Jan 29, 2023

Table of Contents

I. Introduction

The optimization version of the travelling salesman problem (TSP) is a well-known graph problem wherein a salesman needs to compute the most efficient way to visit a set of n cities. Each city must be visited exactly once, and the salesman must end in the same city in which he started. This version of TSP is an NP-hard problem, and the classical brute-force approach requires time (and space) $O(n!)$ for n cities. There are other versions of TSP, and it is often phrased as a decision problem. However, we limit our discussion to the optimization version of TSP as described above and apply a TSP-solver in our pyROUTE library for spatial optimization.

Lots of work has been done in the classical algorithms space to improve upon the naïve, brute force approach to TSP. However, we limit our discussions to those algorithms that contain quantum components.

Rather than analyse and contemplate the diverse array of literature addressing quantum solutions to TSP, our team performed a brief literature review, followed by a critical analysis of the algorithm presented by Srinivasan and colleagues in *Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience* [1] and an implementation of a simplified version of this algorithm.

This report focuses on the quantum TSP solver underpinning `pyROUTE`. For further information on the project, please refer to the project repository.

II. Literature Review

Prior to selecting a specific quantum TSP algorithm to study and implement, we conducted a survey of the existing literature. Rather than addressing all quantum algorithm papers on TPS, this section contains but a brief summary of the works we found most interesting.

A. Goswami et al. (2004)

Goswami and colleagues [2] provide a general framework for efficiently encoding, and subsequently solving, approximate (Euclidean) TSP on a quantum computer. They prove that we can solve approximate TSP in bounded-error quantum polynomial time (BQP) resource bounds if we assume a Gaussian distribution of tour lengths. (Recall BQP is the quantum analogue of the more familiar complexity class bounded-error probabilistic polynomial time (BPP)).

B. Bang et al. (2012)

Bang and colleagues [3] provide a heuristic-based quantum algorithm to solve TSP, wherein a generalized version of Grover's search is used to amplify the probabilities of approximate solutions. It is important to remember that because, in this article, Bang and colleagues discuss heuristic-based algorithms, results are not necessarily globally optimal (heuristic algorithms usually sacrifice optimality for efficiency).

Bang and colleagues prove a quadratic speedup over the quantum algorithm's classical heuristic-based analogue, which basically just comes from the quadratic speedup that Grover provides for comparison-based search. As with [2], this work assumes Gaussian tour costs.

C. Moylett et al. (2017)

Moylett and colleagues [4] prove a quadratic speed up for TSP for bounded-degree (finite) graphs. This is done by applying quantum backtracking to the Xiao-Nagamochi algorithm discussed in [5].

D. Srinivasan et al. (2018)

Srinivasan and colleagues [1] solve TSP by encoding the costs as phases, using phase estimation to obtain the total costs, and using the Dürr and Høyer algorithm [6] to find the minimum cost cycle. This algorithm is integrated into *The Qiskit Textbook* [7] where they note, and improve upon, some circuit errors in the original publication.

III. Algorithm

In [1], Srinivasan and colleagues encode distance costs as phases in a diagonal matrix, and then use pre-identified Hamiltonian cycles to know which eigenstates to use for phase estimation.

We note that pre-identified Hamiltonian circuits allows us to skip phase estimation. If we have access to precomputed Hamiltonian circuits, we can compute the total costs directly.

To this end, we directly compute the total costs from pre-identified Hamiltonian circuit data. At this point, we have an unsorted list of numbers corresponding to total cycle costs. We need to find the minimum of this list, and this is achieved using the Dürr and Høyer algorithm to find the minimum [6] (Grover enhanced binary search).

The Dürr and Høyer algorithm to find the minimum is just binary search on top of Grover's search. The general idea behind Dürr and Høyer is illustrated in Fig. 2.

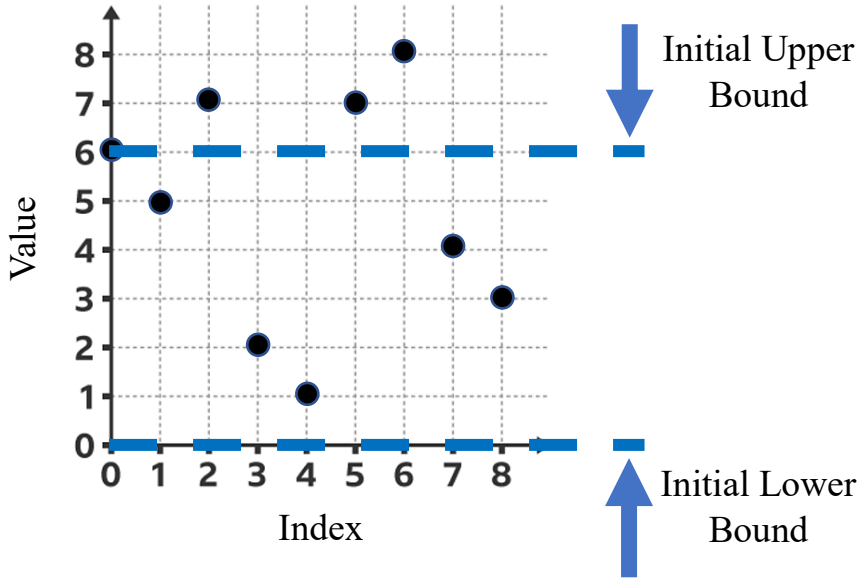

Fig. 1. Visualizing Dürr and Høyer. An example plot of list element value as a function of list index.

Because we are searching a finite list, the function plotted in Fig. 1 has a finitie domain. We are performing binary seach along the y-axis. Because we assumed all distance costs are strictly greater than zero, we can use an initial lower bound of 0. Since we are looking for the minimum, we can just use the first element in our search space as our initial upper bound; worst case scenario, this first element is the largest. At each iteration, we find the midpoint between the bounds, and use Grover to search for an element smaller than said midpoint. If such an element exists, we lower our upper bound to the midpoint, and if no such element

exists, we raise our lower bound to the midpoint. We continue in this fasion till the bounds coverge on the minimum element [6].

For a sorted search space of size M, binary search has the well-known time complexity $O(log(M))$. Here, $M = initial\_upper\_bound - initial\_lower\_bound =$ `arr[0]` where `arr` is the list we are searching.

Grover's search is a quantum algorithm for unstructured search. The quantum circuit for Grover's search is shown in Fig. 2.

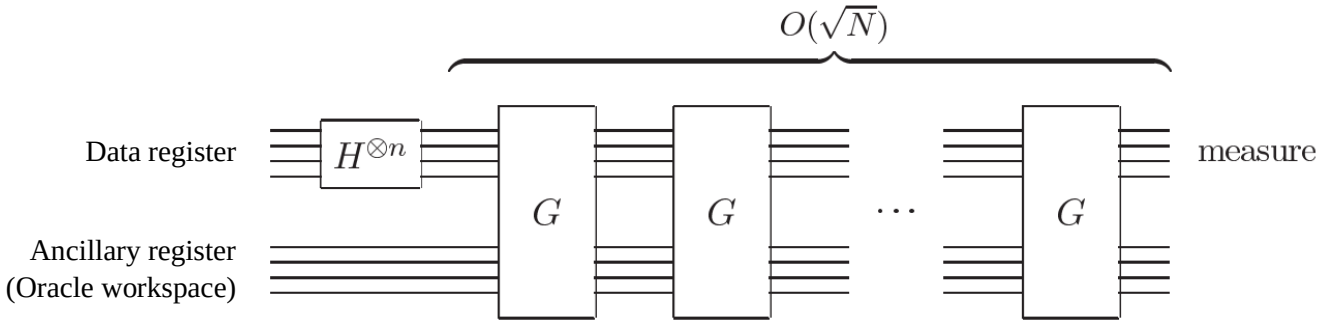

Fig. 2. Circuit for Grover's Search. [8, pg. 251]

Grover's search requires $O(\sqrt{2^n} = \sqrt{N})$ calls to a marking oracle, G, for a search space of n elements [8]. Here, n is the length of our input list `arr`. Combining the complexity of binary search with that of Grover's search, we find the Dürr and Høyer algorithm requires $O(\sqrt{2^n} \cdot log(M))$ queries to G.

Both the algorithm presented by Srinivasan and colleagues in [1] and our adaptation represent a quadratic speedup over the classical brute-force approach. This quadratic speedup aligns with the quadratic speedups featured in our literature review, namely [3] and [4].

IV. Implementation

Our TSP solver was implemented in Python using Qiskit.

TABLE I

SUMMARY OF SOURCE FILES

File	Description of Contents	Quantum or Classical
`grover_enhanced_minimization.py`	The classical binary search overhead for the Dürr and Høyer quantum algorithm to find the minimum.	Completely classical
`grover_for_minimization.py`	The quantum circuit (including the quantum oracle) and the overhead to build and execute the quantum circuit.	Contains both classical and quantum components

Herein, we limit our discussion to the quantum components of the implementation, namely Grover's search. However, the reader is encouraged to explore the classical components of the source code, they are well documented and quite straight forward.

Gover's search was implemented as part of the Dürr and Høyer quantum algorithm for minimization. While the classical binary-search overhead is located in `grover_enhanced_minimization.py`, the quantum subroutine, `grover_for_minimization()`, is located in `grover_for_minimization.py`.

`grover_for_minimization(arr, x)` takes two input parameters, an array of integers `arr` and an integer `x`, and returns **`True`** if a value smaller than `x` exists in `arr`, and **`False`** otherwise. To better understand the problem that `grover_for_minimization()` is trying to solve, and to ensure our binary search overhead was working properly, we first implemented a completely classical function that performs the same task: `grover_for_minimization_classical()`.

Grover's search requires two registers. Our data register contains `len(arr)` qubits. And, for reasons that will become apparent when we discuss our Grover oracle implementation, our ancillary register (the oracle workspace) contains enough ancillary qubits to represent a binary encoding of `sum(arr)` or `x`, whichever is greater. Recall that with n bits you can represent the integers 0 through $2^n - 1$.

At its simplest, Grover's search consists of three steps (built as `circuit`):

1. Create an equal superposition by applying Hadamard gates to each wire in the data register.
2. Use a marking oracle to mark solution states. That is, if the state $|i\rangle$ represents a solution, take $|i\rangle \rightarrow -|i\rangle$. Notice, this phase inversion is only applied to solution states, the oracle leaves all other states untouched.
3. Apply the Gover operator to amplify the probability of the marked elements (amplify the probability of obtaining correct solutions). Optimally, steps 2 and 3 are applied $\left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$ times for a search space of size N containing M marked elements.

It is important to keep in mind that we need to check and see if the element returned from Grover is actually less than `x`. If there are no elements in `arr` less than `x`, no elements will be marked, and Grover will return a random element. If the returned element from Grover is less than `x`, we return **`True`**. Otherwise, we return **`False`**.

1) The Minimization Oracle

One of the most challenging part of this project was designing and implementing the oracle used in step 2 of Grover's search. To understand the chosen approach, consider the example search space $[7, 10, 6, 18]$. Notice, the length of this array is 4, so we will require 4 data qubits. Once we create a superposition, these four data bits together represent 16 binary states:

$$\frac{1}{4}[|0000\rangle + |0001\rangle + \cdots + |1110\rangle + |1111\rangle].$$

As shown in Fig. 4, we perform controlled additions to the ancillary register, controlled off the data register.

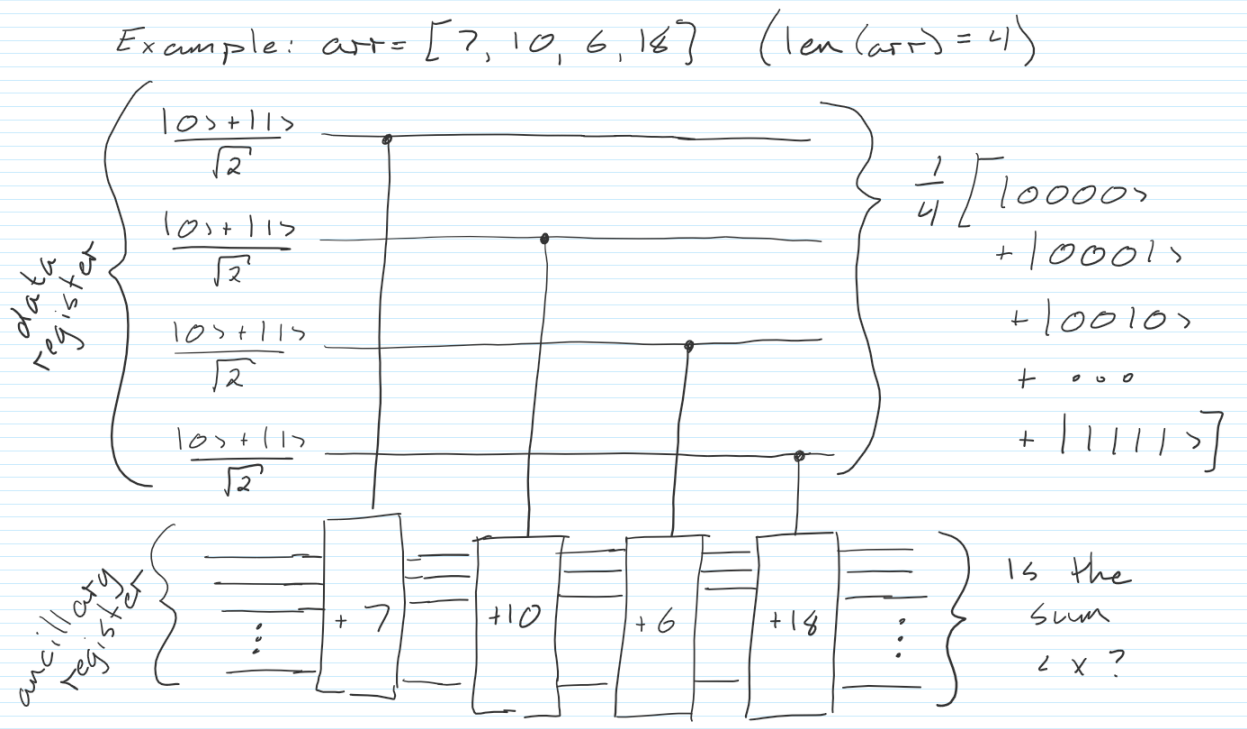

Fig. 3. The Minimization Oracle.

Additions are performed in the Fourier basis using `add_k_fourier()`. We used QFT code adapted from the Qisket Textbook [7] and its adjoint, to convert back and forth between the computational basis and the Fourier basis.

Then, we just check to see if the sum on the ancillary wires is less than `x`. This was done in Pauli primitives. Notice, we need to exclude the |0000⟩ state because 0 will always be less than `x`. This is fine because we assumed all distance costs are strickly greater than zero.

2) Our Circuits

Using `qiskit`, we were able to build up the following circuits from Pauli primatives. Fig.4 shows Grover's search, and Fig. 5 shows our minimization oracle.

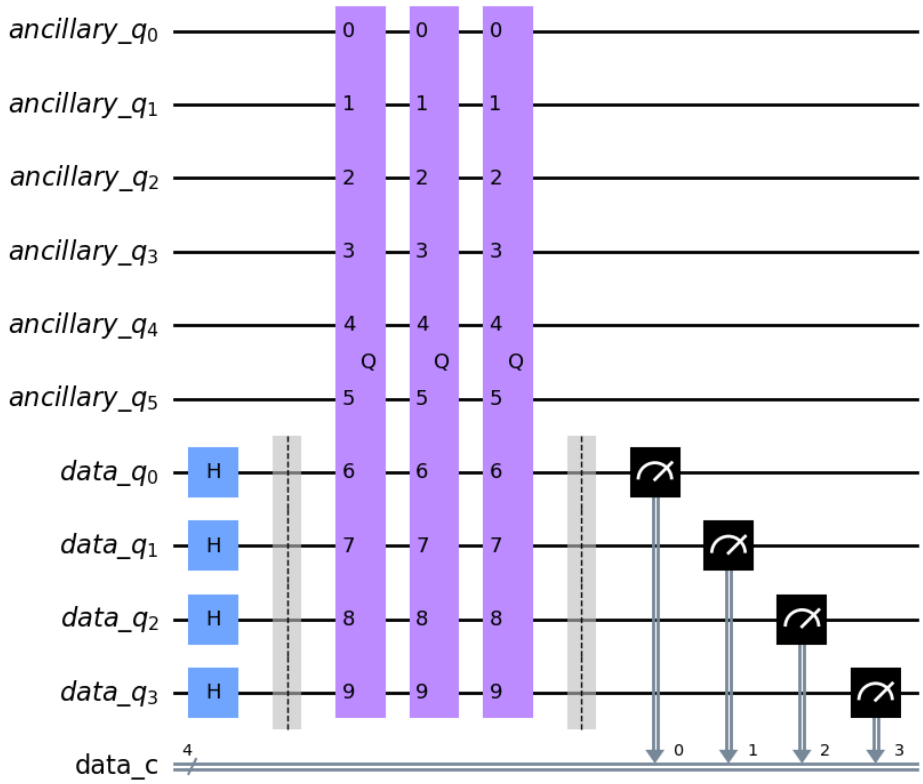

Fig. 4. Our Grover's circuit, built up from Pauli primatives. Notice it is very similar to Fig 2.

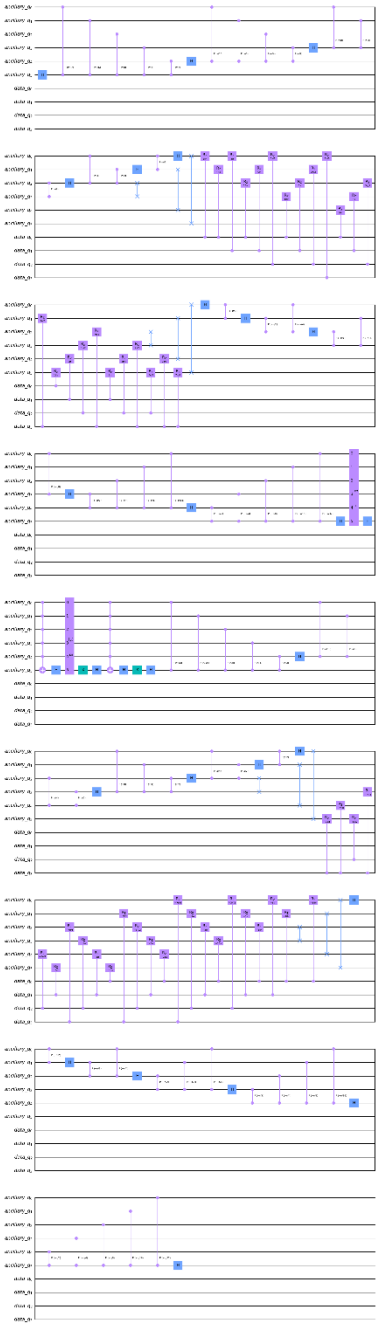

Fig. 5. Our Grover minimization oracle. Details are not important, but notice we have many rotations controlled off the data wires (purple boxes), and these perform the controlled additions illustrated in Fig. 3

V. Conclusion

In this project we implemented a variation of the algorithm for the travelling salesman problem (TSP) presented by Srinivasan and colleagues in [1]. A TSP solver was developed in Python; the quantum components developed using IBM's Qiskit Python library.

In their algorithm, Srinivasan and colleagues encode costs as phases and use phase estimation to extract only those total costs corresponding to pre-identified Hamiltonian circuits. Then, they employ the Dürr and Høyer algorithm to find the minimum cost circuit. We noted that pre-identified Hamiltonian circuits allowed us to compile the list of total costs classically and therefore phase estimation was unnecessary here.

The Dürr and Høyer quantum algorithm for minimization is just binary search on top of Grover's search. Gover's search required a minimization oracle, which was implement by performing controlled additions in Fourier space.

Both the algorithm presented by Srinivasan and colleagues and our adaptation represent a quadratic speedup over the classical brute-force approach. This quadratic speedup aligns with the quadratic speedups featured in the literature.

References

[1] K. Srinivasan, S. Satyajit, B. K. Behera, and P. K. Panigrahi, "Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience," 2018. https://arxiv.org/abs/1805.10928.

[2] D. Goswami, H. Karnick, P. Jain, and H. K. Maji, "Towards Efficiently Solving Quantum Traveling Salesman Problem," 2004, https://arxiv.org/abs/quant-ph/0411013.

[3] J. Bang, J. Ryu, C. Lee, S. Yoo, J. Lim, and J. Lee, "A quantum heuristic algorithm for the traveling salesman problem," *Journal of the Korean Physical Society*, vol. 61, no. 12, pp. 1944–1949, 2012, http://doi.org/10.3938/jkps.61.1944.

[4] D. J. Moylett, N. Linden, and A. Montanaro, "Quantum speedup of the traveling-salesman problem for bounded-degree graphs," *Physical review*. vol. 95, no. 3, 2017, http://doi.org/10.1103/PhysRevA.95.032323.

[5] M. Xiao and H. Nagamochi, "An Exact Algorithm for TSP in Degree-3 Graphs Via Circuit Procedure and Amortization on Connectivity Structure," *Algorithmica*, vol. 74, no. 2, pp. 713–741, 2016, http://doi.org/10.1007/s00453-015-9970-4.

[6] C. Dürr and P. Høyer, "A Quantum Algorithm for Finding the Minimum," 1996. https://arxiv.org/abs/quant-ph/9607014.

[7] IBM, "Solving the Travelling Salesman Problem using Phase Estimation" in *The Quiskit Textbook*. 2021. [Online]. https://qiskit.org/textbook/ch-paper-implementations/tsp.html.

[8] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, 10th anniversary ed. Cambridge: Cambridge University Press, 2010.